PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : G06F 17/30	A2	**(11) International Publication Number:** WO 00/67157 **(43) International Publication Date:** 9 November 2000 (09.11.00)

(21) International Application Number: PCT/US00/11791

(22) International Filing Date: 1 May 2000 (01.05.00)

(30) Priority Data:

60/131,716	30 April 1999 (30.04.99)	US
60/152,521	3 September 1999 (03.09.99)	US
09/480,818	10 January 2000 (10.01.00)	US
09/480,816	10 January 2000 (10.01.00)	US
09/480,319	10 January 2000 (10.01.00)	US
09/481,101	10 January 2000 (10.01.00)	US
09/480,838	10 January 2000 (10.01.00)	US
09/480,834	10 January 2000 (10.01.00)	US
09/480,847	10 January 2000 (10.01.00)	US

(71) Applicant *(for all designated States except US)*: IMAGEX.COM, INC. [US/US]; Suite 200, 10800 NE 8th Street, Bellevue, WA 98004 (US).

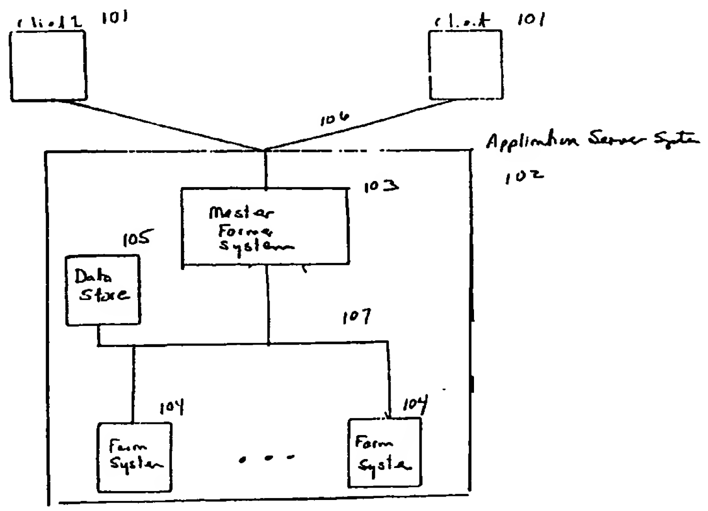
(72) Inventor; and
(75) Inventor/Applicant *(for US only)*: KRUM, Brent [US/US]; Suite 200, 10800 NE 8th Street, Bellevue, WA 98004 (US).

(74) Agents: PIRIO, Maurice, J. et al.; Perkins Coie LLP, Suite 4800, 1201 Third Avenue, Seattle, WA 98101-3099 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD FOR CONFIGURING AN APPLICATION SERVER SYSTEM



(57) Abstract

A method for processing request to service computational tasks. An application server system receives requests to run various jobs. A job indicates that a certain application program is to be executed with a certain set of input. The application server system includes a master computer and multiple slave computers. The master computer receives requests to run jobs, selects a slave computer to run each job, and then assigns each job to slave computer selected for that job. The master computer of the application servers system receives the requests form client computers that may be connected to the application server system via the Internet. A client-side component of the application server system may execute on the client computers to assist users in submitting their requests.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

METHOD FOR CONFIGURING AN APPLICATION SERVER SYSTEM

TECHNICAL FIELD

The present disclosure relates generally to computer systems and, more particularly, to techniques for handling high volumes of processing requests.

BACKGROUND

Many computer systems have been developed to handle high volumes of processing requests from users. In transaction-oriented environments, there is a need to process very high volumes of transactions quickly. Traditionally, such processing requests were handle by a single mainframe computer system. Users of such mainframe computer systems would submit processing requests (*e.g.*, making an airline reservation) from either local or remote terminals. The ability to handle such processing requests or transactions in a timely manner was limited by the resources of the computer system. As a result, mainframe computer systems were designed and developed with ever-increasing amounts of computing resources. For example, the speed of CPUs and the speed and amount of memory has increased dramatically over time. Nevertheless, the actual computing resources of such mainframe computer systems seemed to always lag behind the users needs for computing resources.

Because single mainframe computers were unable to satisfy the users' requirements and because of their high cost, multi-computer systems were developed to help satisfy the requirements. The computing resources of such multi-computer systems can be increased by adding additional computers to the system. Thus, the architectures of the multi-computer systems were in some sense scalable to provide ever-increasing computer resources. A problem with such multi-computer systems has been the high overhead associated with the systems. The high overhead stems from the processing required for the computers to coordinate their activities.

The popularity of the Internet has significantly increased the need for computing resources. A web server may be accessed by thousands and tens of thousands of users each day. These users may require that significant computing resources be expended to satisfy their requests. Current techniques of multi-computer systems are inadequate to handle such a high demand for computing resources. It would be desirable to have a multi-computer system that can handle the increased demand. Also, it would be desirable to reduce the overhead associated with such multi-computer systems.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram illustrating the components of the application server system in one embodiment.

Figure 2 is a block diagram of the components of the master farmer system in one embodiment.

Figure 3 is a block diagram illustrating the components of a farm system in one embodiment.

Figure 4A is a flow diagram of a routine illustrating the processing of an identify farm component of the master farmer system.

Figure 4B is a flow diagram illustrating processing of a routine that calculates an estimated completion time.

Figure 5 is a flow diagram illustrating a routine for calculating a computing resource load for a farm system.

Figure 6 is a flow diagram illustrating the processing of a routine of the field component for starting jobs.

Figure 7 is a flow diagram of an end job routine of the field component.

Figure 8 is a flow diagram of a routine illustrating the processing of a plot component.

Figure 9 is a matrix containing execution times for various combinations of a job sizes and computer resource loads.

Figure 10 is a flow diagram illustrating a routine to update the execution times of the matrix.

Figure 11 is a block diagram illustrating a routine to calculate the computing resource load while a job ran.

Figure 12 is a block diagram illustrating the organization a multiple application server system.

Figure 13 is a block diagram illustrating example components of the farm allocator system.

Figure 14 is a flow diagram illustrating processing of the farm allocator engine.

Figure 15 is a diagram illustrating the process of allocating a farm system to an application server system.

DETAILED DESCRIPTION

A method and system for processing requests to service computational tasks is provided. In one embodiment, an application server system receives requests to run various jobs. A job indicates that a certain application program is to be executed with a certain set of input. The application server system includes a master computer and multiple slave computers. The master computer receives requests to run jobs, selects a slave computer to run each job, and then assigns each job to slave computer selected for that job. The master computer of the application server system receives the requests from client computers that may be connected to the application server system via the Internet. A client-side component of the application server system may execute on the client computers to assist users in submitting their requests. One advantage of this application server system is that slave computers may be dynamically added to or removed from the application server system as the demand for computing resources changes. Another advantage is the very low overhead associated with distributing jobs to slave computers.

The application server system uses various techniques to ensure that jobs are completed as soon as possible given the current load of the application server system. In one embodiment, when the application server system receives a request to run a job, it estimates the time at which each slave computer could complete the job. The master computer then assigns the job to the slave computer that can complete the job the soonest. The application server system may estimate the completion times using actual execution statistics of other jobs for the same application program. By using the actual execution statistics of other jobs, the application server system can estimate completion times that are more accurate and reflect the actual computing resources of the slave computers.

To estimate a completion time for a slave computer, the application server system first determines a start time for the job and then determines the time it will take to run the job on that slave computer. The application server system estimates the time it will take to run the job on a slave computer based on the "size" of the job and the estimated "load" on the slave computer at the time the job would run on that slave computer. Each job is assigned a size that reflects a grouping of jobs that are anticipated to use the same amount of computing resources when they run. Thus, the application server system estimates the execution time of a job based on actual execution statistics for jobs in the same group.

The possible sizes may range from 1 to 100, where the largest possible job for an application program is assigned a size of 100, and the smallest possible job for that application program is assigned a size of 1. Thus, all jobs with the same size (*e.g.*, a size of 20) are assumed to use the same amount of computing resources. Each application is also assigned an "application program load" that reflects an estimate of the computing resources (*e.g.*, percentage of CPU) used by the application program during execution. For example, an application program may be given an application program load of 25 if it typically consumes 25 percent of the computing resources when it is the only application program executing on a slave computer. If each slave computer has the same computing resources (*e.g.*, same number of processors and same processor speed), then one application program load number can be used for each application program on all slave computers. However, if the computing resources of the slave computers differs, then each slave computer could have its own application program load for each application program. The "computing resource load" of a slave computer reflects the amount of computing resources that are being used or would be used when a certain set of application programs are being executed at the same time (or concurrently) on that slave computer. The computing resources load may reflect a percentage of the CPU that is being used. Thus, the computing resource load may range from 1 to 100 percent.

The application server system determines the time it would take to run a job of a slave computer ("execution time") based on review of actual statistics from other jobs for the same application program. In one embodiment, the application server system tracks for each application program the actual execution time of its jobs, the slave computer on which the job ran, and the computing resource load of the slave computer while the job ran. The application server system assumes that similar jobs, that is jobs for the same application program with the same size, will have the same execution time if the job is run in a similar slave environment, that is on the same slave computer with the same computing resource load. Thus, if the actual execution time in a similar slave environment is available for such a similar job, then the application server system uses the actual execution time as an estimate. When such statistics are not available for a similar job that ran in a similar slave environment, the application server system estimates the execution time based on the actual execution time of similar jobs, but in different slave environments. For example, if the actual execution time for a job was 10 seconds with a computing resource load of 50 percent, then the application server system may estimate the execution time of a similar job to be 8 seconds when the computing resource load is 25 percent.

The application server system determines the computing resource load of a slave computer based on the application program loads of the application programs that will be executing at the time. In one embodiment, the application server system identifies which application programs will be executing at the time and then totals the application program loads of the application programs. For example, if 3 jobs will be running at the time and 2 jobs are for an application program with an application program load of 10 and the other job is for an application program with an application program load of 5, then the computing resource load at that time will be 25.

The application server system determines the start time of a job on a slave computer based on the estimated completion times of jobs currently assigned to that slave computer for the same application program. The application server system may track the estimated completion time of each job assigned to a slave computer that has not yet completed. Each slave computer may be configured so that only a certain number of instances of each application program can be executing at the same time. When a slave computer is assigned a job by the master computer, it may queue the assigned job if there is already the configured number of instances of the application program executing. The application server system may determine the start time based on review of the estimated completion times for the currently executing jobs and the queued jobs for that application program. In particular, the start time is the time at which the number of instances would drop below the configured number of instances.

Figure 1 is a block diagram illustrating the components of the application server system in one embodiment. The application server system 102 is connected to client computers ("clients") 101 via communications link 106. A client may be a computer system through which jobs may be submitted to the application server system. For example, a client may be a personal computer with an Internet browser through which a user interacts to select and submit jobs to the application server system. A client may also be an automated system that interacts directly with the application server system. The client may include a client-side component of the application server system. The client-side component may assist the user in defining a job (*e.g.*, selecting an application and specifying the input and output files) and submitting the job. The client-side component may also provide the user with updates on the progress of the jobs that have been submitted. The communications link may be the Internet, local area network, a dial-up link, or any other communications link. In the following, an agricultural metaphor is used to describe the components of the application server system. The application server system includes a master farmer system 103, farm systems 104, and a data store 105, which are connected via communications link 107. The master farmer system (*e.g.*, a master computer) receives requests to submit jobs from clients, identifies a farm system (*e.g.*, slave computer) to run the job, and instructs the identified farm system to run the job. When a farm system receives an instruction to run a job, it queues the job until an instance of the application program is available to run that job. When the job runs, it retrieves input data from and stores output data in the data store. The data store may be a file system, database management system, or other storage system.

The computers of the application server system and the clients may be computing devices that include one or more central processing units, memory, and various input/output devices. The memory may include primary and secondary storage and other computer-readable media. The modules, routines, components, and so on of the application server system may be implemented as a computer program comprising executable instructions. The computer programs and data structures of the application server system may be stored on computer-readable medium. The computer-readable medium may include a data transmission medium (*e.g.*, the Internet) for transmitting the computer programs, data structures, and inter-computer communications from one computer to another. The application programs served by the application server system may be any type of computer program.

Figure 2 is a block diagram of the components of the master farmer system in one embodiment. The master farmer system 200 includes a job entry component 201, a job database 202, a distribute jobs component 203, an identify farm component 204, a collect statistics component 205, a job statistics database 206, an update status component 207, a farmer connection component 208, and a farmer data base 209. These components and databases represent the functions performed by and data used by the master farmer system. The job entry component interfaces with the clients to receive request to run jobs. The job entry component may provide a client with a list of available application programs and input files. The client may submit a job, which is a combination of an application program and input files, to be run. When the job entry component receives the submission of the job, it may update the jobs database and provide the job to the distribute jobs component. The distribute jobs component is responsible for identifying to which farm system a job should be assigned. The distribute jobs component invokes the identify farm component to identify the farm system to which the job should be assigned. The identify farm component may select a farm system based on characteristics (*e.g.*, size) of the job and statistics in the job statistics database that relate to the similar jobs or may rely on information provided by the slave computers. Once the identify farm component selects a farm system, the distribute jobs component notifies the identified farm system that the job has been assigned to it. The collect statistics component retrieves statistical information on the execution of jobs from the farm system and then updates the job statistics database. For example, the job statistics database may contain information describing each job, the job size, the farm system on which it ran, the actual execution time of the job, and the computing resource load on the farm system at the time the job ran. The identify farm component may use these and other statistics to identifying the farm system to which a job is to be assigned. Alternatively, the identify farm component may provide each farm system with an indication of the job and request it to provide an estimate of the completion time. The update status component monitors the running of jobs and provides update information on the progress of the jobs to the clients. The farm connection component receives requests from farm systems to establish or break a connection with the master farmer system. The farm connection component may also monitor the farm systems to determine whether a connection with a farm system has been broken because the farm system has failed. When a connection is broken, the farm connection component may direct the distribute jobs component to assign the uncompleted jobs of that farm system to other farm systems. The farm connection component maintains the farm database which contains information describing the farm systems that are currently connected to the master farm system. The farm database may contain an identification of each farm system (*e.g.*, address) along with an indication of the application programs that each farm system can execute.

Figure 3 is a block diagram illustrating the components of a farm system in one embodiment. The farm system 300 includes a farm module 301 and a field component 302 for each application program that the farm system can execute. When the farm system is assigned a job by the master farmer system, it queues the job to be run by the appropriate field component. Each field component has a job queue 305 associated with it. In one embodiment, the field components execute within the same process as the farm module. For example, each field component may be a dynamic link library that is automatically loaded into the same process as the farm module when the farm module is launched. When a farm module is launched, it may check which field components to load. Each field component may be customized to watch and monitor jobs for a particular application program. Each field component may be configured to have at most a certain number of instances of its application program executing concurrently. When the field component detects that a job is in its queue, it determines whether that configured number of instances is currently executing. If not, the field component launches a plot component to launch and monitor an instance of the application program for the job. The plot component detects when the instance is no longer executing and notifies the field component. The field component may collect statistics about the running of each job and store the statistics in the field statistics database 307. The farm module may periodically supply statistics to the master farmer system. When the master farm system is identifying a farm system to assign to a job, the master farm system may request a farm system to estimate the completion time of that job if assigned to it. The farm system uses the completion time estimator 306 to generate the estimate.

Figure 4A is a flow diagram of a routine illustrating the processing of an identify farm component of the master farmer system. The identify farm component selects the farm system to which a job is to be assigned. In one embodiment, the routine selects the farm system that is estimated to complete the job soonest. The master farmer system may maintain sufficient information to calculate an estimated completion time for each farm system. Alternatively, the master farm system may provide each farm system with an indication of a job and request the farm system to provide an estimated completion time. The flow diagram of Figure 4A illustrates the processing of the routine for a master farm system that requests each farm system to supply an estimated completion time. In blocks 401-406, the routine loops selecting each farm system to which the job can be assigned. A job can be assigned to any farm system that can execute the application program of the job. In block 401, the routine selects the next farm system to which the job can be assigned. The farm database may contain a mapping from each farm system to the application programs that it is configured to execute. In decision block 402, if all the farm systems have already been selected, then the routine is done, else the routine continues at block 403. In block 403, the routine requests the selected farm system to provide an estimated completion time for the job. In decision block 404, if the completion time is less than the minimum completion time estimated for any previously selected farm system, then the routine continues at block 405, else the routine loops to block 401 to select the next farm system. In block 405, the routine sets the minimum completion time to the estimated completion time for the selected farm system. In block 406, the routine identifies the selected farm system as having the soonest completion time. The routine then loops to block 401 to select the next farm system.

Figure 4B is a flow diagram illustrating processing of a routine that calculates an estimated completion time. This routine is invoked by a field component after the farm module of the farm system receives a request to provide an estimated completion time. The routine is passed the size of the job. In block 410, the routine calculates the start time for the job. The start time may be initially calculated based on the completion times of the jobs currently assigned to the field component. The initial start time may be calculated by identifying when a plot component will be first available for running the job. A plot component will first be available depending on the number of plot components configured for the field component. For example, if the field component has been allocated five plot components, then the initial start time will be the fifth latest completion time of the job assigned to the field component. That is, when the job with the fifth latest completion time completes, then a plot component will be available to assign to the job. However, the job may not necessarily be able to start at that time. In particular, if the application program load would cause the farm system to exceed its maximum computing resource load at some point during its execution, then the farm system may decide that the job should not be started at that time. In such a case, the routine may return an indication that the job cannot be assigned to the farm system. Alternatively, the routine may analyze the jobs assigned to each field component to determine the earliest time when the job can execute without causing the farm system to exceed its maximum computing resource load and return that time as the start time. In block 411, the routine invokes a calculate computing resource load routine to calculate the computing resource load on the farm system at the start time. In block 412, the routine adds the load to the calculated computing resource load to give the estimated computing resource load for the farm system while the job is running. In decision block 413, if the computing resource load is greater than 100, then the routine returns an indication that the job cannot be assigned to this farm system, else the routine continues at block 414. In block 414, the routine calculates the execution time for the job based on the estimated computing resource load and the job size. Various techniques for calculating the estimated execution time are described below. In block 415, the routine calculates the estimated completion time by adding the start time to the execution time. The routine then completes.

Figure 5 is a flow diagram illustrating a routine for calculating a computing resource load for a farm system. This routine is passed an indication of a time. The routine calculates the computing resource load to be the total of the application program loads of each job that is scheduled to be executing at that time. In blocks 501-507, the routine loops selecting each field component of the farm system and adding the application program loads for the jobs of that field component that will be executing at the time. In block 501, the routine selects the next field component of the farm system. In decision block 502, if all the field components have already been selected, then the routine returns, else the routine continues at block 503. In blocks 503-507, the routine loops selecting each job currently assigned to the selected field component. A job is currently assigned to a field component if it is currently running or queued up to run in that field component. In block 503, the routine selects the next job of the selected field component. In decision block 504, if all the jobs have already been selected, then the routine loops to block 501 to select the next field component of the farm system; else the routine continues at block 505. In block 505, the routine retrieves the start time and completion time of the selected job. The start and completion times may be stored in a data structure associated with the job. In decision block 506, if the selected job will be executing at that passed time, then the routine continues at block 507, else the routine loops to block 503 to select the next job. In block 507, the routine adds the application program load of the selected job to the computing resource load for the farm system and then loops to block 503 to select the next job.

Figure 6 is a flow diagram illustrating the processing of a routine of the field component for starting jobs. This routine may loop waiting for a plot component to become available. When a plot component becomes available, the routine then retrieves the next job from the queue and assigns it to a plot component. In decision block 601, if a plot component is currently available, then the routine continues at block 602, else the routine loops to wait for an available plot component. Alternatively, this routine may be invoked whenever a plot component becomes available or whenever a new job is placed in the queue. If either a job is not in the queue or a plot component is not available, then the routine would return without waiting. In block 602, the routine retrieves the next job from the queue. If there is no job in the queue, then the routine waits until a job is placed in the queue. In block 603, the routine assigns the job to an available plot component. In block 604, the routine collects and saves pre-execution statistics relating to the job. For example, the routine may store the job size, the start time of the execution, and the current computing resource load of the farm system. In block 605, the routine launches the job. The routine may launch a job by launching a plot component to run in a process separate from the process of the farm module. The routine passes an indication of the job to the plot component. The plot component starts an instance of the application program passing an indication of the input. In block 606, the routine notifies the farm module that the job has begun its execution. The routine then loops to block 601 to assign a job to the next available plot.

Figure 7 is a flow diagram of an end job routine of the field component. The end job routine is invoked whenever a plot component indicates that it is ending its execution. A plot component may end its execution for various reasons including normal or abnormal termination of the application program. In block 701, the routine unassigns the job from the plot component, which makes the plot component available to execute another job. In block 702, the routine collects and saves post-execution statistics relating to the job. The statistics may include the time at which the job completed. In block 703, the routine notifies the farm module that the job has completed. The routine then completes.

Figure 8 is a flow diagram of a routine illustrating the processing of a plot component. A plot component may be a custom program for controlling the execution of a certain application program. The input to this routine may be an indication of the input and output files for the application program. In block 801, the routine launches an instance of the application program. The application program may be launched by creating a new process for executing the application and passing an indication of the input and the output to that process. Alternatively, the application program may be launched by creating a thread of execution within the same process in which the plot component is executing. The application program may be a dynamic link library that is loaded by the plot component. In blocks 802-806, the routine loops processing events relating to the execution of the application program. A plot component may be notified of certain events relating to the application program. For example, the application program may notify the plot component that it is terminating normally or that it has detected a failure. The plot component may also be notified when the application program terminates as a result of a programming error (*e.g.*, causing a system fault). In one embodiment, the plot component may periodically check the output file of the application program to determine whether it is being modified. If the output file is not modified for certain period of time, then the plot component may assume that the application program is not functioning as expected. In block 802, the routine waits for notification of an event. In decision blocks 803-806, the routine determines whether a termination event has occurred. If so, the routine continues at block 807, else the routine loops back to wait for the next event. In block 807, if the application program is still executing, the routine forces the instance of the application program to abort. The routine then reports that the job has ended to the field component along with the reason for ending.

Figure 9 is a matrix containing execution times for various combinations of a job sizes and computer resource loads. Each field component may have such an execution time matrix associated with it. The field components use this matrix when estimating execution times of application programs. The matrix includes a row for each group of job sizes. In this example, job sizes 1-10 are grouped together, job sizes 11-20 are grouped together, and so on. The matrix includes a column for each grouping of computing resource loads. In this example, computing resource loads 1-10 are grouped together, computing resource loads 11-20 are grouped together, and so on. One skilled in the art will appreciate that different groupings of the job sizes and computer resource loads may be used depending on the desired accuracy of execution times. For example, the computing resource loads may be grouped into 100 groups to have a more refined execution time. When the farm system needs to determine an estimated execution time for a job with a job size of 15 running with a computing resource load of 25, the farm system retrieves the execution time of .77 from the row corresponding to the job sizes 11-20 and the column corresponding to the computing resource load of 21-30.

In one embodiment, a field component updates the estimated execution times in the matrix whenever a job completes. The field component calculates the actual execution time of the job based on the actual start time and completion time for the job. The field component then calculates the computing resource load of the farm system during execution of that job. The farm system may periodically retrieve and store the actual computing resource load of the farm system. Some computer systems may specify that certain registers will contain various indications of computing resource loads. For example, one computer system may continually update a register to indicate the percentage of the CPU that was used over a very short interval. The farm module may use this percentage as the actual computing resource load. To estimate the computing resource load while a job ran, the field component averages actual computing resource loads recorded by the farm module while the job ran.

The field component, in one embodiment, updates the matrix by replacing an entire row whenever a job completes. For example, when a job of size 15 completes, then the field component updates each execution time in the row corresponding to job sizes 11-20. The field component sets the estimated execution time for that job size and the calculated computing resource load to the actual execution time of the job. For example, if the job size was 15, the calculated computer resource load was 25, and the actual execution time was .82, then the field component would replace the .77 of the matrix with .82. The field component may also project that .82 execution time to the other computing resource loads for that job size. For example, the field component may set the execution time for the computing resource load of 1-10 to .81 and the execution time for the computing resource load of 21-30 to .83. In general, the field component will have a projection factor that is used to project the actual execution time for job of the certain size throughout the various computing resource loads for that size. The field component may have a single projection factor for the entire matrix or may have different factors for each job size group. These projection factors can be empirically derived by running sample jobs of each job size group on farm systems with different computing resource loads. Alternatively, rather than simply replacing the execution times of a row based only on the single actual execution time, the field component may also factor in the existing execution in a row. For example, the field component may average the new execution time with the execution time (projected or actual) currently stored in the matrix. The field component may also use any type of function, such as exponential decay, to combine the new execution time with previous execution times. Also, the field component may alternatively or in combination adjust the execution times for other job sizes. The matrices may also be maintained on the master farmer system if the master farmer system is to calculate the completion times.

Figure 10 is a flow diagram illustrating a routine to update the execution times of the matrix. This routine is passed a job size, a computing resource load, and an actual execution time. In block 1001, the routine retrieves the projection factor for the matrix. Although the projection in this example is linear, one skilled in the art will appreciate that nonlinear projections may also be used. In blocks 1003-1006, the routine loops setting the execution time for each entry in a row. In block 1003, the routine selects the next computing resource load group. In decision block 1004, if all the computing resource loads have already been selected, then the routine is done, else the routine continues at block 1005. In block 1005, the routine increments an index to count the number of computer resource load groups. In block 1006, the routine sets the execution times in the matrix to the projected execution time. The routine calculates the actual execution time (e) as follows:

$$e \square k \square (i \square j) \square f \square k$$

where k is the actual execution time, i is the column number of the selected computing resource load, j is the index of the passed computing resource load, and f

is the factor. The routine then loops to block 1003 to select the next computing resource load group.

Figure 11 is a block diagram illustrating a routine to calculate the computing resource load while a job ran. The routine is passed the actual start time and actual completion time of the job. The routine calculates the average of the actual computing resource loads during running of the job. In block 1101, the routine retrieves and totals the actual computing resource load recordings made by the farm system between the start time and the completion time. In block 1102, the routine sets the computing resource load to that total divided by the number of readings. The routine then returns.

The application server system calculates the size of each job assuming that jobs of approximately the same size will generally use the same amount of computing system resources when they run. In one embodiment, the application server system calculates the size of a job based on the sizes of the input files and output files of a job. Certain application programs may use an amount of computing resources that varies in relation to be size of the input files. While other application programs may use an amount of computing resources that varies in relation to the size of the output files. Some application programs may even use an amount of computing resources that varies in relation to a combination of the sizes of the input and output files. Such application programs whose size can be based on the corresponding size of the input or output files may be CPU intensive application programs. To base the size of job on the size of the input file, the application server system may calculate the size of the job to be the percentage of the size of the input file to the maximum size of an input file for the application program. For example, if the maximum size input file is 600 MB and the input file size for the job is 300 MB, then the size of the job would be 50 percent (*i.e.*, 300/600). If the actual input file size happens to be larger than 600 MB, then the job size is set to 100.

From the time an initial start time, execution time, and completion time are estimated until the current time, the actual computing resource load of the farm system to which a job is assigned may make these estimated times inaccurate.

In particular, as other jobs are run, a more accurate picture of these times may become available. As a result, a re-estimation of the start time, execution time, and completion time may be more accurate than the initial estimation. For example, the actual start time may be earlier or later and, as a result, the execution time may be different depending on the computing resource load at that time. During the actual running of the job, the computing resource load of the farm system may also vary considerably from the estimated computing resource load used to estimate the execution time. In one embodiment, the master farmer system may periodically request a farm system to re-estimation the completion time of its jobs. The re-estimation of the completion time of jobs that have not yet been started may be done using the same technique as when the completion time was initially estimated when determining which farm system to assign the job. The same technique, however, would not necessarily yield accurate times if the job was already running. If a job is already running, then the farm system may re-estimation the completion time by estimating the percentage of the job that is not yet complete and estimating the computing resource load of the farm system during the completion of the job. The farm system can then estimate the execution time for the job if it were to start at the current time. This estimation can be done using the same technique as used to initially estimate execution times, which is based on the job size and computing resource load. The farm system can then multiply the estimated percentage by the estimated execution time to give the estimated time to completion.

The farm system may use various techniques to determine the percentage of a job that is yet to be completed. In one embodiment, each application program as it executes may report its estimate of the percentage of completion. Alternatively, a plot component can monitor the input activity or output activity of the application program. The plot component may be able to estimate the size of the output file based on the size of the input file. If so, the plot component can monitor the size of the output file and base its estimate on the size. The plot component may also be able to monitor the requests of the application program to read from the input file. The percentage of the input file that has been read by the application program may indicate the percentage that is complete.

The re-estimation of completion times can be initiated based on various events. The re-estimation can be done periodically and reported to the clients who submitted the jobs at any time. The clients may also submit a request to re-estimation the completion time of its jobs. The master farmer system or a farm system may detect that a certain job will not complete at the time estimated. For example, the estimated completion time may have already past or the estimated start time may have already past. Also, it may be detected that the estimated completion times of other jobs are not consistent with the actual completion times. In such situations, a re-estimation of the completion times may also be initiated.

In one embodiment, multiple application server systems may be used to meet the processing needs of clients. For example, application server systems may be geographically distributed throughout the world to meet the processing needs of clients throughout the world. The processing requests of clients may be sent to the geographically closest application server system. Such an organization of application server systems may be particularly useful in the Internet environment. Although the placing of application server systems at various geographic locations may speed the communications between clients and the application server systems, it may create bottlenecks in certain application server systems. For example, one application server system at a certain to geographic location may not have sufficient farm systems at that location to satisfy the clients in its geographic area. Whereas, another application server system may have a surplus of farm systems. Moreover, the overall load on an application server system may vary considerably overtime. For example, the overall load on application server systems may be high during business hours and low during non-business hours of that geographic area. In one embodiment, a farm allocator system is used to dynamically change the allocation of farm systems of the various application server systems. As a result, some farm systems may not be geographically local to the master farmer system to which they are allocated. In addition, each farm system notifies its master farmer system when it is to be allocated or deallocated. (Thus, a slave computer is responsible for notifying its master computer when it is to be and not be its slave.)

Figure 12 is a block diagram illustrating the organization a multiple application server system. The multiple application server system includes a farm allocator system 1201, application server systems 1202, and unallocated farm systems 1203. The farm allocator system receives statistical information from the application server systems. The statistical information may indicate the overall load of the application server systems and may indicate the characteristics of the jobs that are running on the application server systems. The farm allocator system decides when to allocate farm systems to and deallocate farm systems from the application server systems. When the farm allocator system decides to allocate a farm system to an application server system, it notifies the farm system of the master farmer system and provides information for configuring the farm system. The configuration information may include, for example, which application programs that the farm system is to support. The communications link between the farm allocator system and the master farmer systems and farm systems may be the Internet. The farm allocator system may provide the farm system with the Internet protocol ("IP") address of the master farmer system to which it is allocated. The farm system then notifies the master farmer system that it has been allocated to it and provides it with the farm system's configuration information. The master farmer system then updates its farm database. The master farmer system may then reassign jobs that are currently assigned to other of its farm systems to the newly allocated farm system. Alternatively, the master farmer system may wait until it receives a new request to run a job from a client and assign that job to the newly allocated farm system in normal course. The deallocation of a farm system from a master farmer system works in an analogous manner. In particular, the farm allocator system notifies the farm system to be deallocated. That farm system in turn notifies its master former system, which updates its farm database so that no more jobs will be assigned to that deallocated farm system. The master farmer system may also reassign those jobs currently assigned to the deallocated farm system to other farm systems.

Figure 13 is a block diagram illustrating example components of the farm allocator system. The farm allocator system 1300 comprises a farm allocator engine 1301, an application server configuration database 1302, an application server statistics database 1303, an unallocated farm system database 1304, an application server interface 1305, and a farm system interface 1306. The farm allocator engine identifies when farm systems should be allocated to and deallocated from application server systems based on the information in the configuration database and statistics database. The configuration database contains information describing the current configuration of the application server systems. The configuration information indicates which farm systems are currently allocated to each application server system and indicates the number of each application program that a farm system is configured to execute. The statistics database contains statistical information about the running of jobs on each application server system. The statistical information may include historical data on the execution times, submission times, start times, and so on for each job that ran on a farm system. The application server interface receives the statistical information from the application server systems and stores that information in the statistics database. The farm system interface controls the notifying of the farm systems of their allocation and deallocation. The unallocated farm system database identifies those farm systems that are unallocated and contains information describing the characteristics of the farm systems. The characteristics may include the processing power, location, and the application programs that can be executed by the farm system.

Figure 14 is a flow diagram illustrating processing of the farm allocator engine. The farm allocator engine may be periodically invoked to determine whether farm systems should be allocated or deallocated. In block 1401, the engine selects the next application server system. In decision block 1402, if all the application server systems have already been selected, then the engine is done, else the engine continues at block 1403. In block 1403, the engine calculates the overall load on the selected application server system. The engine may also have access to historical information to predict the overall load so that farm systems can be allocated in deallocated in advance as needed. In decision block 1404, if the overall load used too high, then the engine continues at block 1405, else the engine continues at block 1406. In block 1405, the engine allocates a farm system to the selected application server system. The engine may select a farm system that is geographically close to the application server system. Alternatively, as part of this allocation process, the engine may deallocate a farm system from another application server system so that it can be allocated to the selected application server system. The engine then loops to block 1401 to select the next application server system. In decision block 1406, if the overall load of the selected application server system is too low, then the engine continues at block 1407, else the engine loops to block 1401 to select the next application server system. In block 1407, the engine notifies a farm system of the selected application server system to deallocate. The engine may select the farm system to deallocate based on the characteristics (e.g., location) of the farm system. The engine then loops to block 1401 to select the next application server system.

Figure 15 is a diagram illustrating the process of allocating a farm system to an application server system. Flow diagram 1500 represents the processing of the farm allocator system, flow diagram 1510 represents the processing of a farm system, and flow diagram 1520 represents the processing of a master farmer system. The processing of flow diagram 1500 is invoked to allocate a farm system to an application server system. In block 1501, the farm allocator system sends a notification to the selected farm system. This notification includes information identifying the master farmer system to which the farm system is to be allocated and may include configuration information for the farm system. In block 1510, the farm system receives the notification. In block 1511, the farm system configures itself based on the configuration information. In block 1512, the farm system notifies the master farmer system to which it is allocated. In block 1520, the master farmer system receives the notification. In block 1522, the master farmer system updates its farm database to reflect the allocation. The master farmer system may also reassign jobs to the farm system at that time. In block 1523, the master farmer system sends a confirmation to the farm system. In block 1513, the farm system receives the confirmation and sends the confirmation to the farm allocator system. In block 1502, the farm allocator system receives the confirmation and updates its configuration and unallocated farm system databases as appropriate.

The farm allocator system may dynamically configure farm systems as they are allocated to application server systems. The farm allocator system may determine which application programs should be executed by the farm system to be allocated. The farm allocator system may make this determination based on the statistics in the application server statistics database. For example, if a certain application server system cannot service in a timely manner all the jobs for a certain application program, the farm allocator system may configure a farm system to be allocated or already allocated to run jobs only for that certain application program. In one embodiment, the farm allocator system may direct the storing in a file system directory of the farm system one or more files containing the field component and plot component corresponding each application program that the farm system is configured to execute. The farm system can then look to that directory to identify which application programs it may execute. The directory may also contain the actual application program. The field component may be stored as a dynamic link library, and the plot component may be stored as an executable file. During its operation, the farm system will look to that directory to determine for which application programs it may run jobs. In one embodiment, the configuration of a farm system can be changed dynamically by adding files to or removing files from the directory after startup of the farm system. Whenever a farm system receives a request to estimate the completion time of a job to be assigned, the farm system may check the directory and only provide an estimate if the directory indicates that the farm system can run that job. The farm module of the farm may link in field components dynamically as the corresponding dynamic link library file is detected in the directory.

The application server system may monitor its various components to identify when a failure occurs. A failure may be a failure of hardware of farm system; failure of the operating system of the farm system; failure of a farm module, a field component, or a plot component; failure of an application program; failure as a result of bad input data; and so on. Certain of these failures may be catastrophic in the sense that the master farmer system immediately knows that it should not assign any additional jobs to the farm system and may even reassign those jobs currently assigned to the farm system. For example, a hardware failure, such as a memory failure, may make it impractical for jobs to run on the farm system. Other failures may not be catastrophic in the sense that the farm system may continue to operate, albeit in a diminished capacity. If the capacity is so diminished, the master farmer system may want to stop assigning jobs to that farm system.

The various components of the farm system may monitor the execution of other components to detect failures. In particular, the farm module may monitor the execution of the field components, the field components may monitor the execution of the plot components, the plot components may monitor the execution of the application programs, and an application program may monitor its own execution. Whenever a failure is detected, the monitoring component is notified of the failure and the reason for the failure. The monitoring component may then in turn notified its monitoring component. Ultimately, a client may be notified of certain failures. However, each monitoring component may take steps to restart the activity that failed. For example, if the farm module detects that a certain field component has failed, that farm module may then re-queue the jobs that were assigned to that field component and then restart that field component. If another failure is detected, then the farm module may notify the master farmer system that jobs for the application programs corresponding to the failed field component can no longer be run. Similarly, if a plot component detects that an application program has failed, it passes on the reason for the failure to the field component. A plot component may detect that an application program has failed by monitoring the input file access, output file access, or both of the application program. If, for example, the application program has not accessed these files or not added data to the output file, it may be assumed that the application program has failed. Each monitoring component may decide whether to notify its monitoring component or attempt to correct the failure itself. The various components may fail as a result of programming errors (*e.g.*, resulting in a general protection fault) or as a result of hardware errors.

A master farmer system may notify the farm allocator system when a failure has occurred at a farm system. The farm allocator system may want to allocate an additional farm system to that application server or to reconfigure a farm system currently allocated to that application server system. Alternatively, the master farmer system may direct the reconfiguration of one of its farm systems.

Based upon the above description, it will be appreciated that, although specific embodiments of the invention have been described herein for purposes of illustration, various modifications may be made without deviating from the spirit and scope of the invention. Accordingly, the invention is not limited except as by the appended.

CLAIMS

1. A method in an application server system for handling a failure in a farm system, the method comprising:

receiving notification of a failure in the farm system;

identifying all the jobs that were currently assigned to the failed farm system; and

for each identified job,

selecting a farm system that can complete the identified job the soonest; and

assigning the identified job to the selected farm system.

2. The method of claim 1 wherein the failure is a hardware failure of the farm system.

3. The method of claim 1 wherein the failure is detected as a result of the farm system not responding.

4. The method of claim 1 wherein the farm system indicates to a master farmer system that a failure has occurred.

5. The method of claim 1 wherein the failure is a failure of a field component.

6. The method of claim 1 wherein the failure is a failure of a plot component.

7. A method in an application server system for processing failures, the method comprising:

under control of a plot component of a farm system,

detecting that an application program for a job has failed; and

notifying a master farmer system of the farm system that failure of the application program has been detected; and

under control of the master farmer system,

when the failure is not a result of improper input to the application program, assigning the job to an other farm system; and

when the failure is a result of improper input to the application, notifying a client that submitted the job.

8. The method of claim 7 including sending to the client that submitted the job of a new estimated completion time.

9. The method of claim 7 wherein the notification to the client indicates a reason for the failure.

10. The method of claim 7 wherein the detecting that an application program for a job has failed includes monitoring the size of an output file of the application program.

11. The method of claim 7 wherein the detecting that an application program for a job has failed includes monitoring reading of an input file of the application program.

12. A method in a computer system for selecting a computing device for running a job, the job having a job size, the method comprising:

for each of a plurality of computing devices, determining a start time of the job if the job were to be run on the computing device;

determining a computing resource load on the computing device if the job were to start at the start time;

determining an execution time of the job on the computing device based on the job size and computing resource load of the computing device at the start time; and

determining a completion time based the start time and execution time; and

selecting the computing device with the earliest completion time.

13. The method of claim 12 wherein each job has an application program with an application program load and the computing resource load is calculated based on the application program loads of the jobs currently executing on the computing device.

14. The method of claim 13 wherein the application program load is a measure of a central processing unit used by the application program while executing on the computing device.

15. The method of claim 14 wherein the measure is a percentage of central processing unit used by the application program.

16. The method of claim 12 wherein the job size of the job is based on size of a file processed by the job.

17. The method of claim 16 wherein the file is an input file.

18. The method of claim 16 wherein the file is an output file.

19. The method of claim 12 wherein each job has an application program with an application program load and the determining of a start time includes analyzing application programs executing or scheduled to execute on the computing device to determine a time when the application program load of the application program can be accommodated by the computing device.

20. The method of claim 12 wherein the determining of the execution time includes analyzing information from previous executions of jobs on the computing resource.

21. A method in a computer system for determining an execution time for a job to run a computing device, the method comprising:

determining a job size of the job;

determining a computing resource load for the computing device when the job is to run; and

estimating the execution time for the job to run on the computing device, the estimating based on analysis of the actual execution time of another job, the analysis factoring in the determined job size, the job size of the other job, and a computing resource load of a computing device when the other job executed.

22. The method of claim 21 including generating a mapping of execution times to various combinations of job sizes and computing resource loads.

23. The method of claim 22 wherein the mapping is based on actual execution times of jobs.

24. The method of claim 21 wherein the analysis factors in actual execution time of a job of the same job size running on a computing device with a

computing resource load that is different from the determined computing resource load.

25. The method of claim 21 wherein the analysis factors in actual execution time of multiple jobs of approximately the same job size that run with various computing resource loads.

26. A method in a computer system for generating an estimate for the execution of a job, the job having a job size, the method comprising:

determining a percentage of the job that is yet to complete;

determining a computing resource load of a computing device to which the job is assigned;

determining an execution time based on the job size and computer resource load; and

determining a completion time based on the current time plus the determined percentage of the determined execution time.

27. The method of claim 26 wherein the estimate is generated in response to detecting that a running job will not complete by an estimated completion time.

28. The method of claim 26 wherein the estimate is generated in response to a user query.

29. The method of claim 26 wherein the estimate is generated periodically.

30. The method of claim 26 wherein the job has not yet started.

31. The method of claim 26 wherein the job has started.

32. A method in a computer system for establishing a connection between a master resource and a slave resource, the master resource for directing operation of the slave resource, the method comprising:

determining that a slave resource should connect to a master resource;

notifying the slave resource that it should connect to the master resource;

under control of the slave resource,

upon receiving notification that it should connect to the master resource, sending to the master resource notification that the slave resource is to connect to the master resource; and

under control of the master resource,

upon receiving the notification sent from the slave resource, directing the operation of the slave resource.

33. The method of claim 32 including under control of the slave resource, notifying the master resource to disconnect from the slave resource.

34. The method of claim 32 wherein the master resource and the slave resources are part of an application server system.

35. A method in slave resource for connecting to a master resource, the method comprising:

detecting notification to connect to the master resource;

upon detecting the notification, notifying the master resource to control the operation of the slave resource; and

receiving operating instructions from the master resource and performing the instructed operations.

36. The method of claim 35 including notifying the master resource to stop controlling the operation of the slave resource.

37. A method in a farm system for organizing field and plot component, the method comprising:

starting a farm module to execute in a first process, the farm module having a dynamic link library for each field component, each field component having an associated application program;

under control of the started farm module, invoking a field component to start the running of a job for the application program;

under control of the invoked field component, starting a plot component to execute in another process; and

under control of the started plot component, starting the execution of the application program.

38. The method of claim 37 wherein the starting of the execution of the application program starts the execution in a process different from the process of the plot component.

39. The method of claim 37 wherein the starting of the execution of the application program starts the execution in the same process as the plot component.

40. A method in a computer system for monitoring the running of jobs, each job having an associated application program, the method comprising:

for each job,

starting execution of an instance of a plot component in one process, the plot component for monitoring the execution of an application program; and under control of the plot component,

starting execution of an instance of the associated application program in another process; and

monitoring the execution of the executing application program whereby every executing plot component and every executing application program has its own process.

41. The method of claim 40 wherein a field component for an application program starts the execution of the instances of the plot component for that application program and wherein the field component monitors the execution of the plot component.

42. A method for configuring a farm system of an application server system, the method comprising:

for each of a plurality of application programs, storing in a directory of a file system of the farm system files containing a field and a plot component for the application program; and

during startup of the farm system,

locating the files of the directory; and

configuring the farm system to execute the application programs represented by the field and plot components.

43. The method of claim 42 wherein a farm allocator system directs the storing of the field and plot components in the directory when the farm system is to be allocated to the application server system.

44. The method of claim 42 wherein the field component is a dynamic link library.

45. The method of claim 42 wherein the plot component is stored in an executable file.

46. The method of claim 42 including storing other files containing a field and a plot component for an application program in the directory after startup of the farm system, wherein the farm system is re-configured to execute that application program.

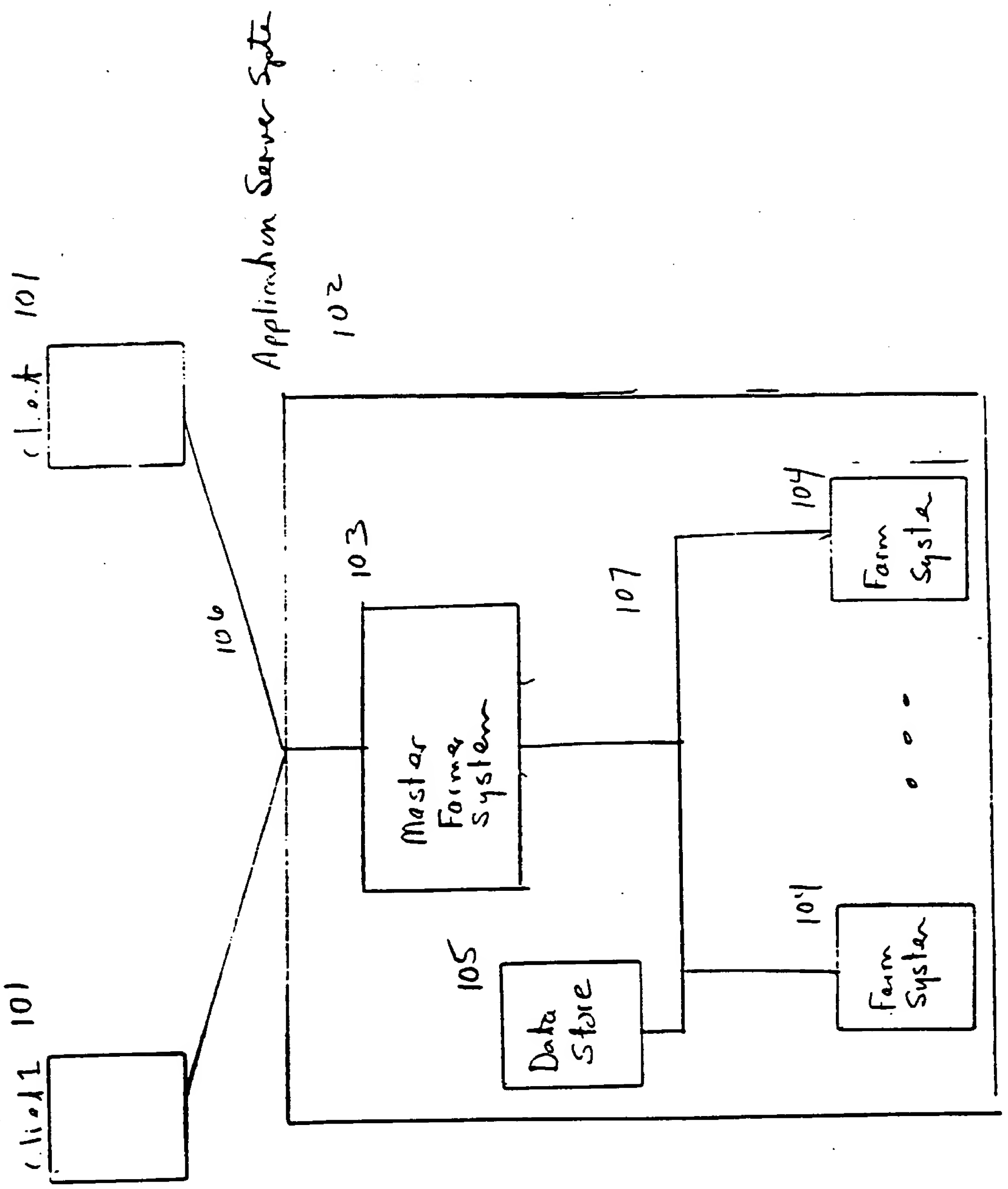

client 101
client 101
Application Server System 102
106
103
Master Farm system
107
104
Farm System
105
Data Store
104
Farm System

Fig 1

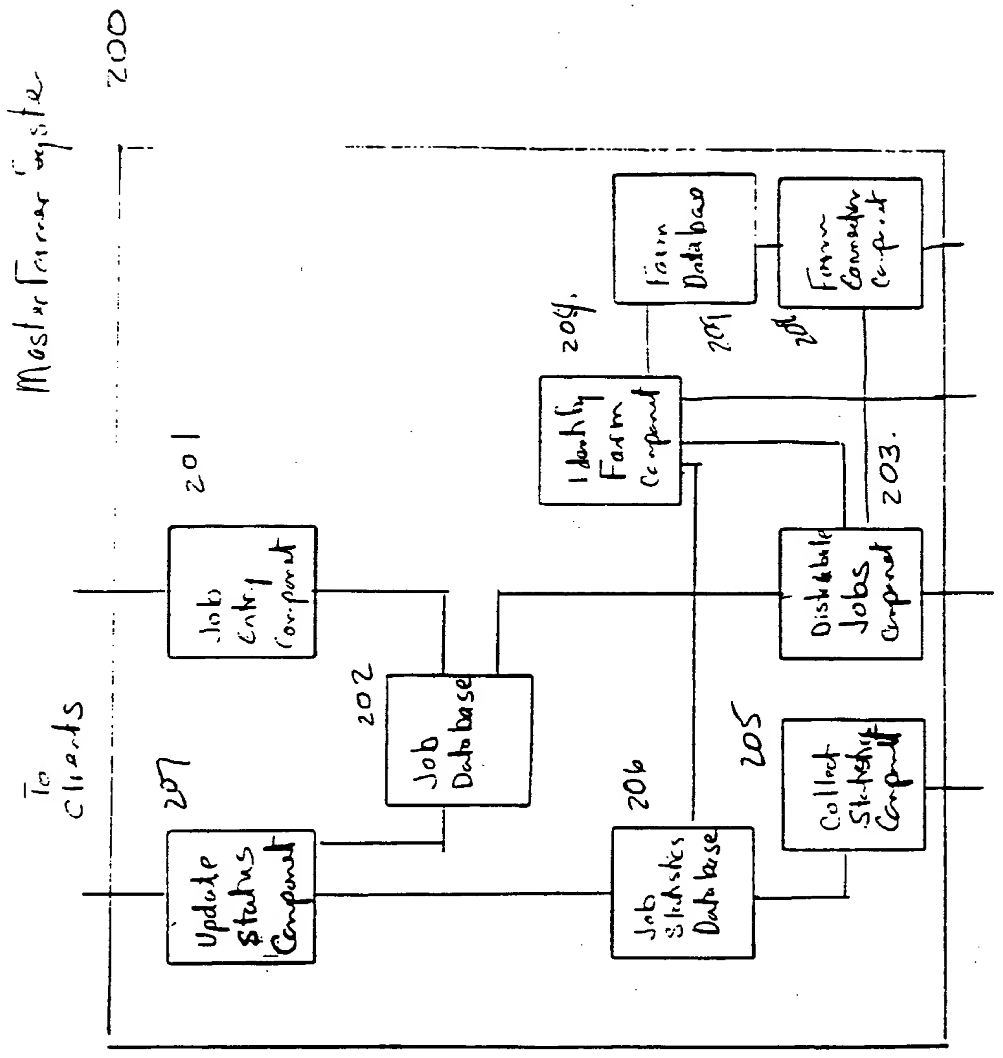

Master Farmer System
200
To Clients
201
Job Entry Component
202
Job Database
203
Distribute Jobs Component
204
Identify Farm Component
205
Collect Statistics Component
206
Job Statistics Database
207
Update Status Component
209
Farm Database
208
Farm Connector Component
To Farm Systems

Fig 2

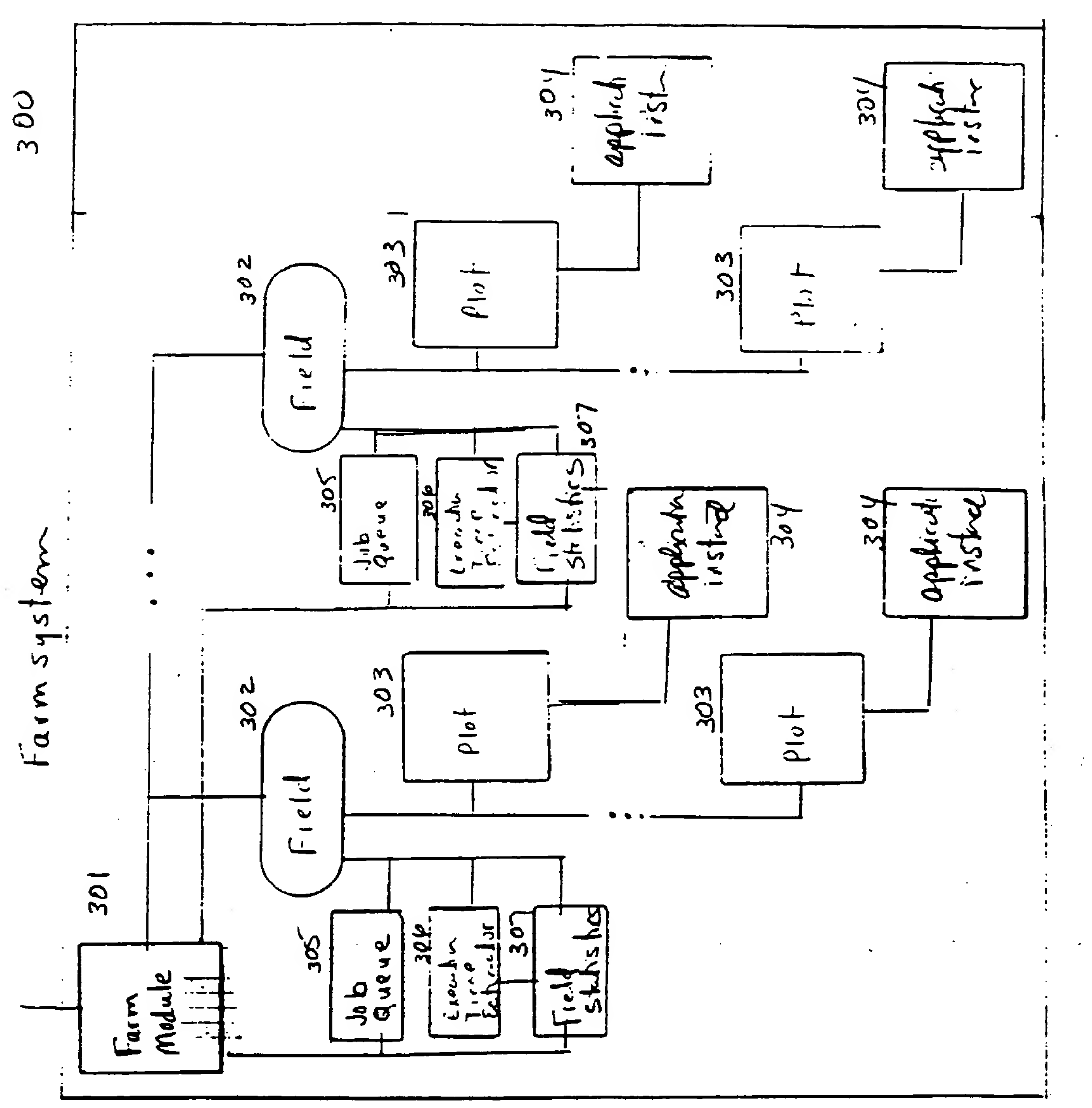

Farm system
300
Farm Module
301
Field
302
Plot
303
applicat instr
304
Job Queue
305
Growth Time Extractor
306
Field statistics
307

Fig 3

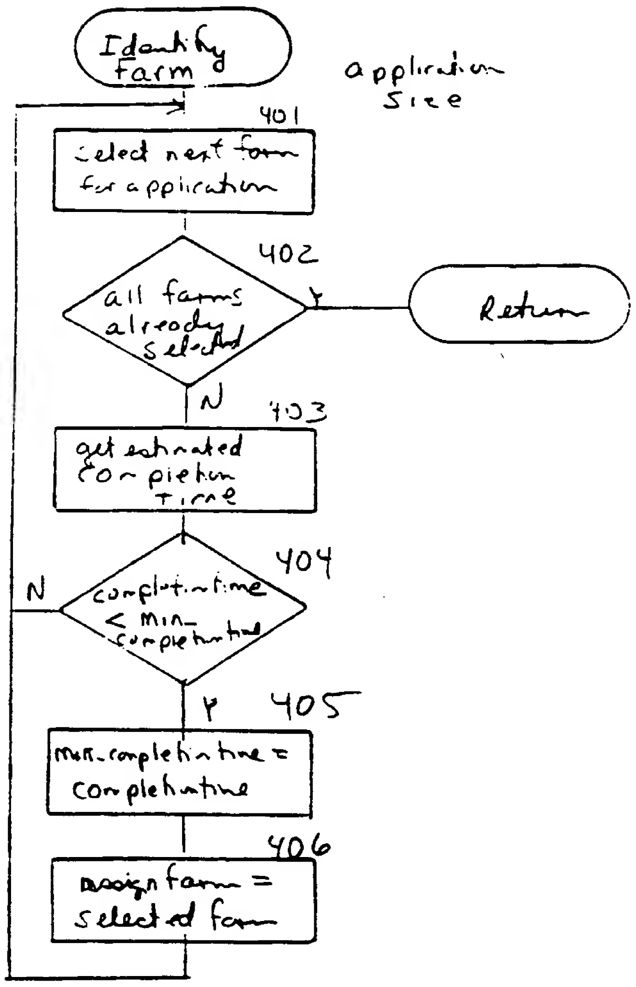

Identify farm
application size
401
select next farm for application
402
all farms already selected
Y
Return
N
403
get estimated completion time
404
N
completion time < min_completion time
Y
405
min_completion time = completion time
406
assign farm = selected farm

Fig 4A

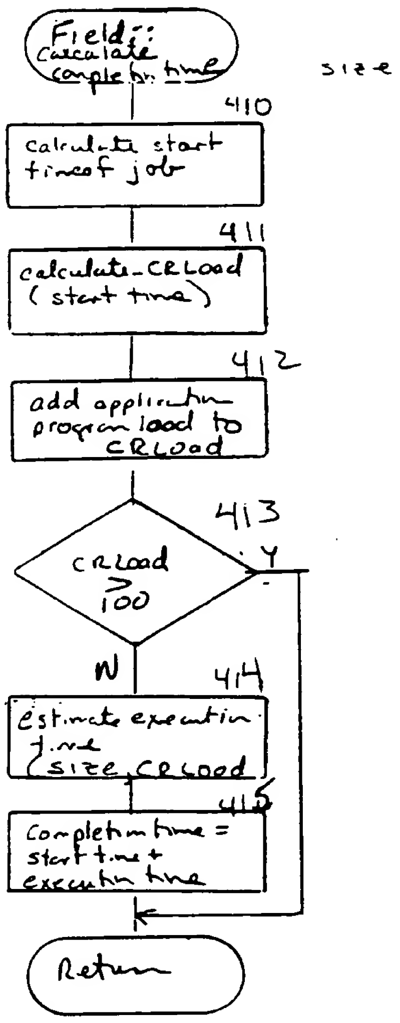

Field: Calculate completion time
size
410
calculate start time of job
411
calculate_CRLoad (start time)
412
add application program load to CRLoad
413
CRLoad ≥ 100
Y
N
414
estimate execution time (size, CRLoad)
415
Completion time = start time + execution time
Return

Fig 4B

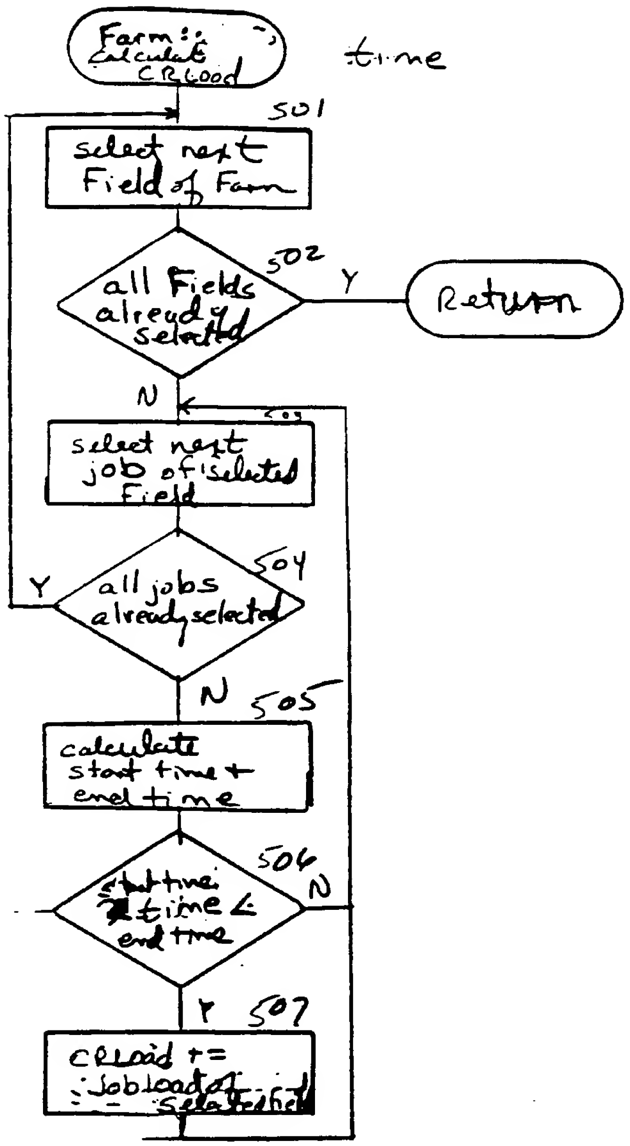

Farm:: Calculate CRLoad
time
501
select next Field of Farm
502
all Fields already selected
Y
Return
N
503
select next job of selected Field
504
Y
all jobs already selected
N
505
calculate start time + end time
506
start time ≤ time ≤ end time
N
Y
507
CRLoad += JobLoad of selected field

Fig 5

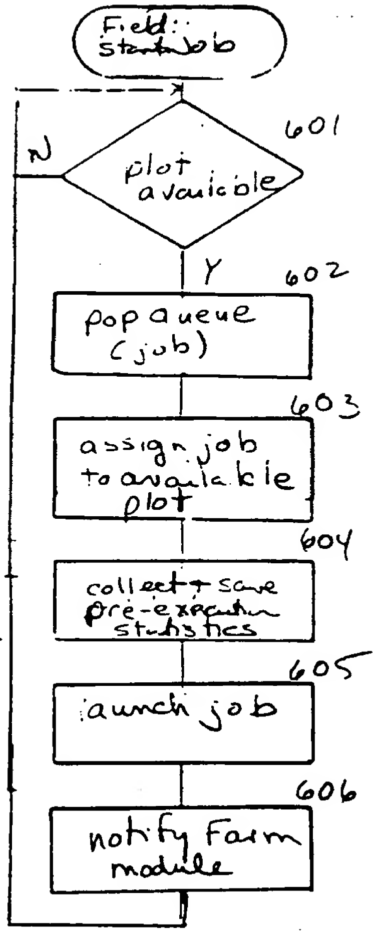

Field:: startJob
601
plot available
N
Y
602
pop queue (job)
603
assign job to available plot
604
collect + save pre-execution statistics
605
launch job
606
notify Farm module

Fig 6

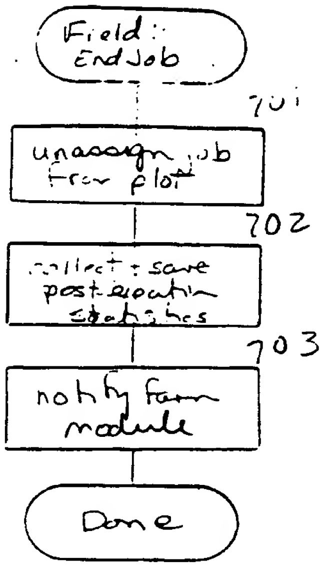

Field: EndJob
701
unassign job from plot
702
collect & save post-execution statistics
703
notify farm module
Done

Fig 7

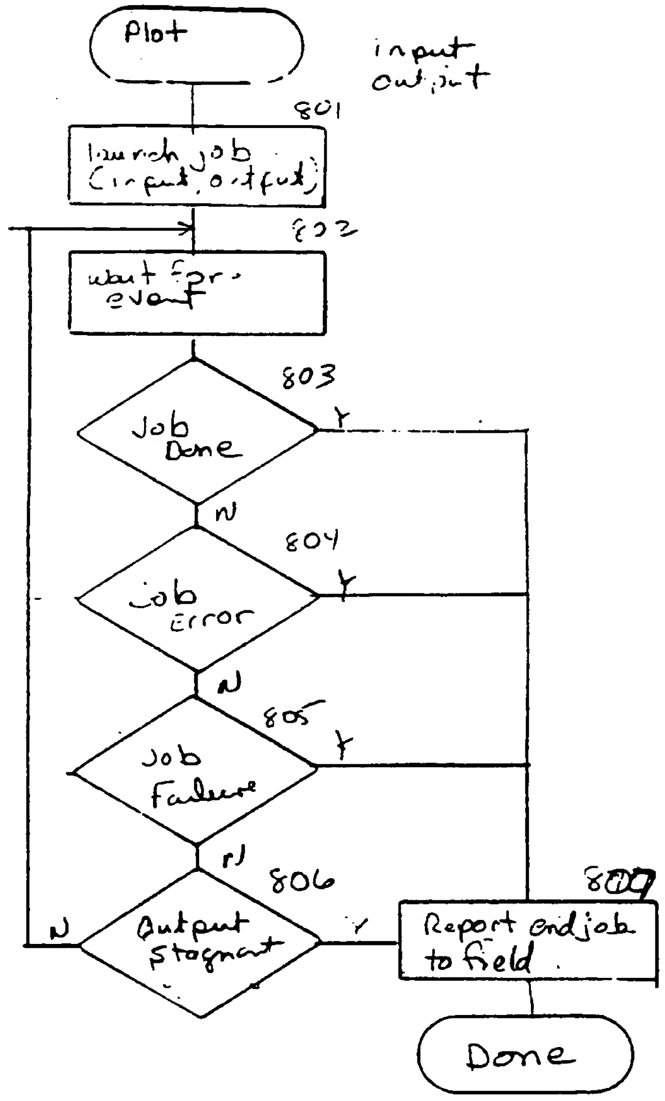

Plot
input
output
801
launch job (input, output)
802
wait for event
803
Job Done
Y
N
804
Job Error
Y
N
805
Job Failure
Y
N
806
Output Stagnant
N
Y
807
Report end job to field
Done

Fig 8

Estimated execution times

Computer Resource Load

Job size	1-10	11-20	21-30	. . .	91-100
1-10	.49	.5	.51		.58
11-20	.75	.76	.77		
91-100	3.2	3.3	3.4		4.1

Fig 9

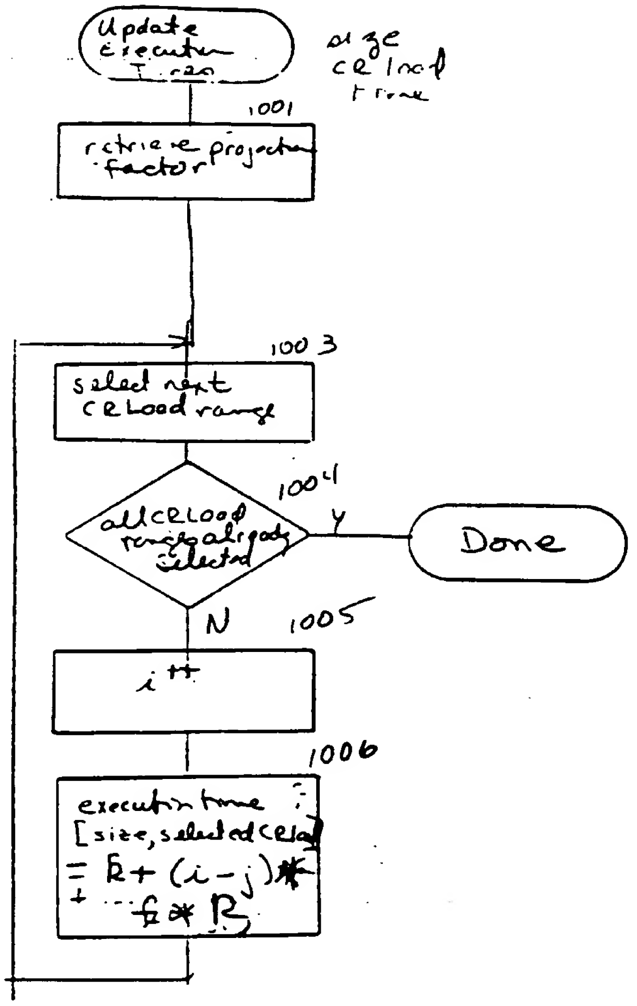

Update execution time
size celoop range
1001
retrieve projection factor
1003
select next celoop range
1004
all celoop ranges already selected
Y
Done
N
1005
i++
1006
execution time [size, selected celoop]
= R + (i - j) *
+ ... f * R

Fig 10

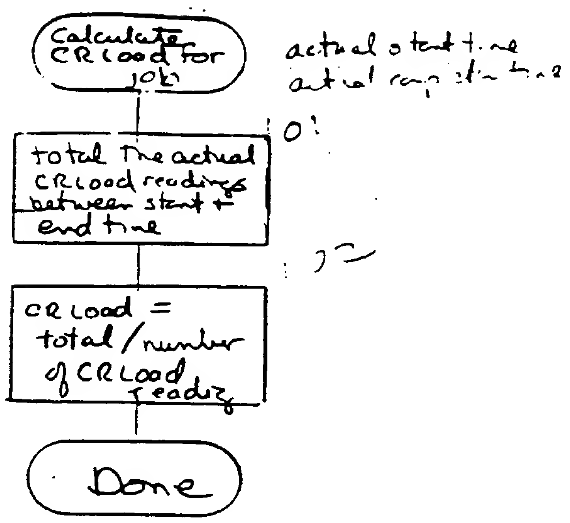

Calculate CRLoad for job
actual start time
actual completion time
total the actual CRLoad readings between start + end time
CRLoad = total / number of CRLoad readings
Done

Fig 11

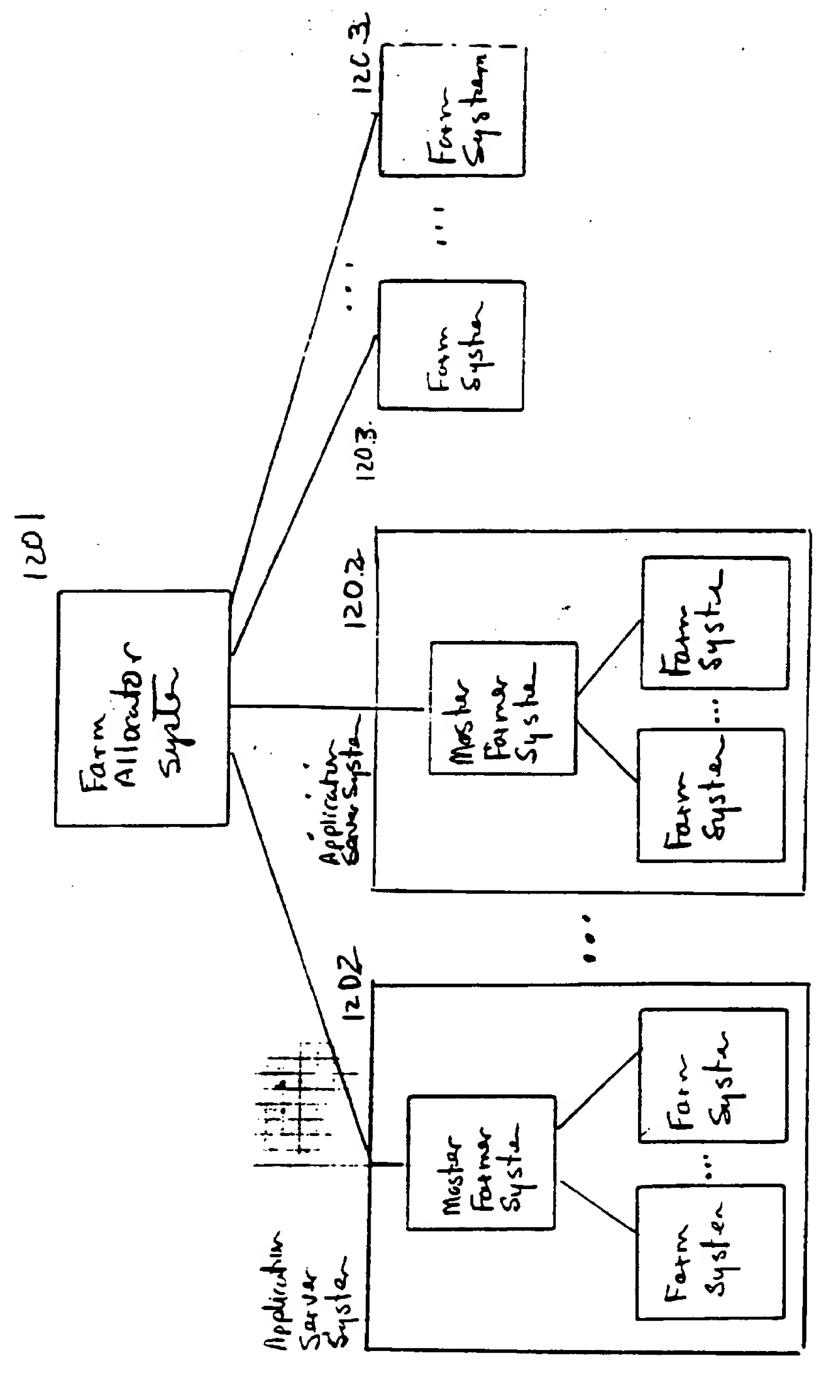

1201
Farm Allocator System
1202
Application Server System
Master Farmer System
Farm System
Farm System
120.2
Application Server System
Master Farmer System
Farm System
Farm System
120.3
Farm System
120.3
Farm System

Fig 12

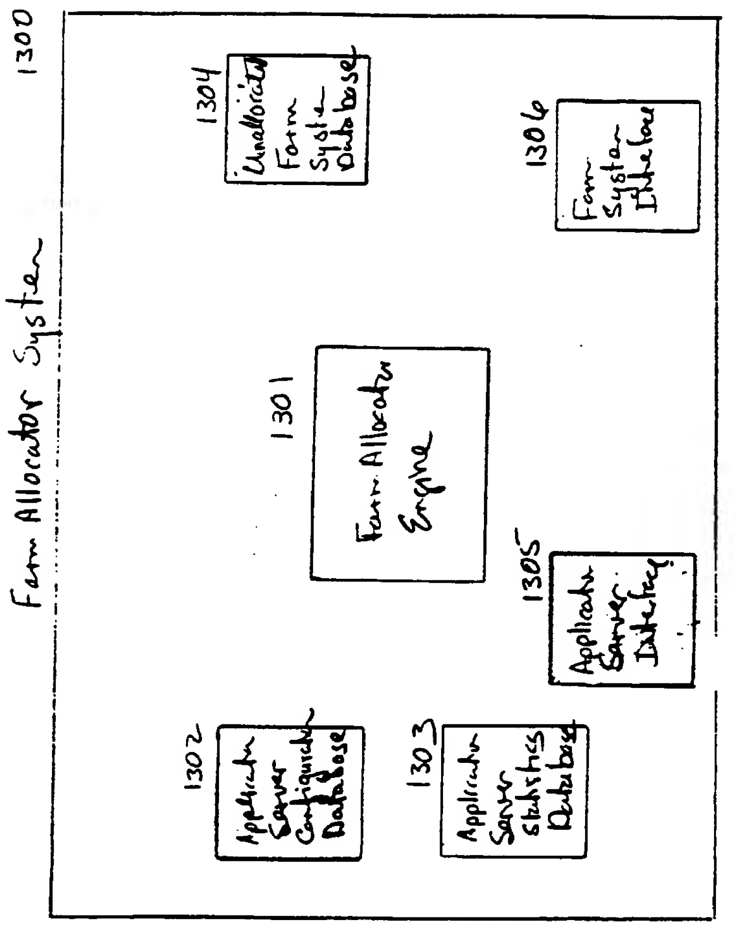

Farm Allocator System
1300
1301
Farm Allocator Engine
1302
Application Server Configuration Database
1303
Application Server Statistics Database
1304
Unallocated Farm System Database
1305
Application Server Interface
1306
Farm System Interface

Fig 13

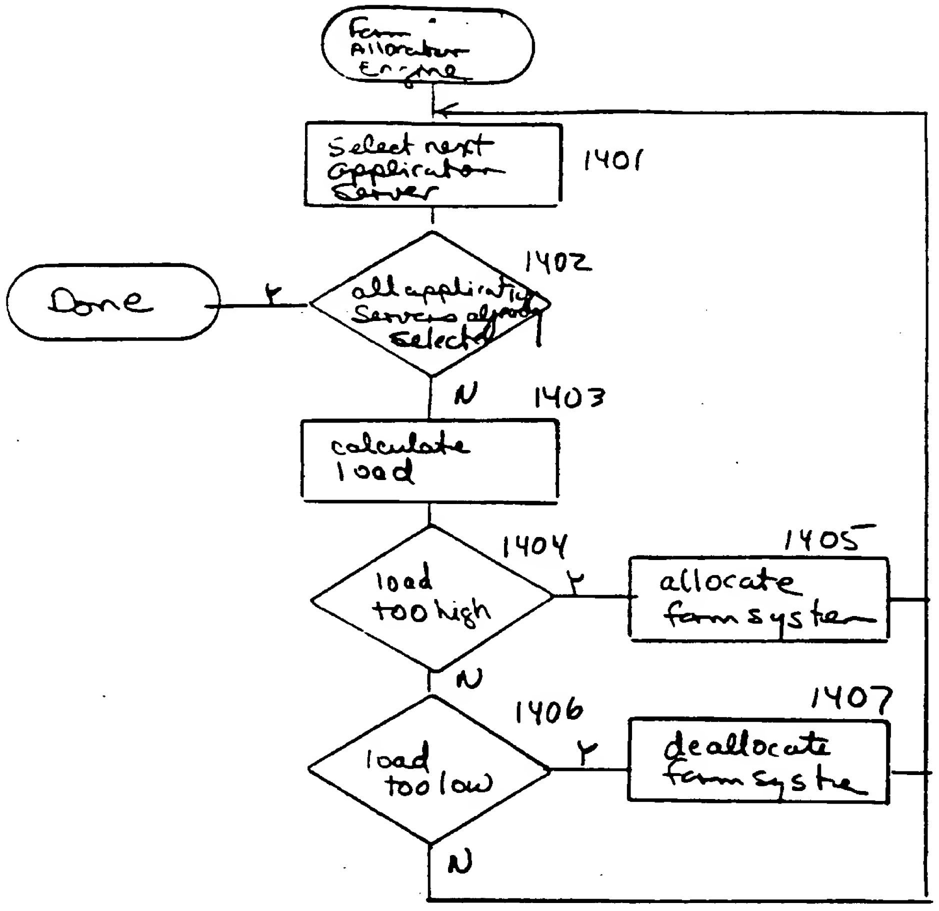

Farm Allocator Engine
Select next application server
1401
all application servers already selected
1402
Y
Done
N
1403
calculate load
load too high
1404
Y
allocate farm system
1405
N
load too low
1406
Y
deallocate farm system
1407
N

Fig 14

Allorate Farm Syste

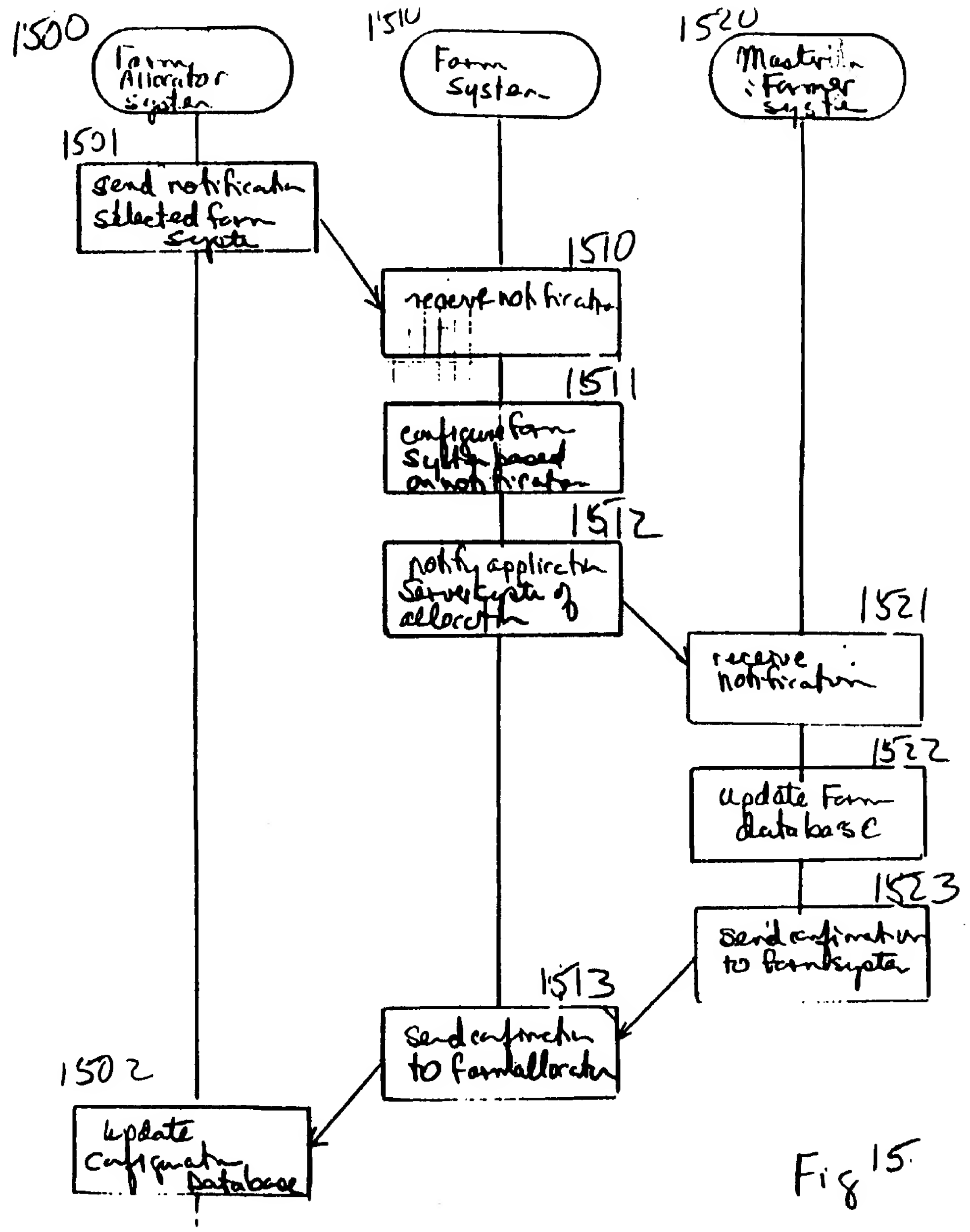

1500
Farm Allocator system
1510
Farm System
1520
Master Farmer system
1501
send notification selected farm system
1510
receive notification
1511
configure farm system based on notification
1512
notify application server system of allocation
1521
receive notification
1522
update Farm database
1523
send confirmation to farm system
1513
send confirmation to farm allocator
1502
update configuration database

Fig 15